System Test Plan

(TINF18C, SWE I, II Praxisprojekt 2019/2020)

Projekt: AMLEngine-DLL Interfaces

Kunde : Rentschler & Ewertz
Rotebühlplatz 41
70178 Stuttgart

Lieferant: Team 4 Joshua, Kevin, Krister, Lucas, Markus, Robin
Rotebühlplatz 41
70178 Stuttgart

Version	Datum	Autor	Kommentar
0.1	13.04.2020	Kevin	Dokument erstellt
0.2	19.04.2020	Kevin	Erweiterung der Test Cases
0.3	29.04.2020	Kevin	Behebung von Fehlern
0.4	1.05.20	Kevin	Updaten mancher Testdaten
1.0	5.05.20	Kevin	Fertigstellung

Table of Contents

1. Scope

The STP (System Test Plan) specifies the test strategy and test planning. It references tests to be performed to verify the accordance of the demanded features given by the SRS (System Requirements Specification) to the implemented features. The document derived from the STP is the STR (System Test Report) where additionally the results are given.

2. Definitions

TC	Testcase
CLI	Command User Interface
AML	AutomationML
AMLX	AutomationMLContainer
DLL	Dynamic Linked Library

3. Product names and attributes

The following test objects must be verified:

Ref.-Id.	Product Number	Product Name	Product Description
1	Version 1.0	C++ Wrapper	Compiler settings to allow the usage of the AMLEngine.dll in a C++ project
2	Build 1.0	AML Console Application	CLI for validating and (de-)compressing of AML Files
3	Build 1.0	Javascript Wrapper	Wrapper for AmlEngine.dll in Node JS

4. Features

Req. - ID	Functionality	Priority	Testsuite ID
LF10	C++ Functions	C	TS-001
LF20	C++ usability	C	TS-001
LF30	Javascript Functions	A	TS-002
LF40	Javascript Usability	A	TS-002
LF50	Import	A	TS-003
LF60	(De-)Compression	B	TS-004
LF70	Validation	B	TS-005

5. Test preparation strategy

The test schematics is based on the project structure. A test suite for each individual use-case is created. They each contain several tests. In this project there are main test suits for the JS Wrapper, C++ Wrapper and the Console Application.

The JS Wrapper has to provide the AML functions. Each function has to validate the user input. Also there has to be checked whether all data needed is delivered.

The C++ Wrapper has to enable the developer to use the Amlengine.dll in his project. It has to be tested for full functionality and usability.

6. Test execution strategy

Since this is a test for a new software product, a full test is required. This test should be divided in the following parts:

6.1. Javascript Wrapper
6.2. Console Application
6.3. C++ Wrapper

Since the Javascript Wrapper is the most complex part, it should be tested as early as possible.

After that the Console Application shall be tested, as it is the next product in regard of complexity.

In the last step, the C++ Wrapper shall be tested.

7. Testing setup

7.1. Console Application

The only test equipment required is a fully functional version of the product.

7.2. C++ Wrapper

The only test equipment required is a fully functional version of the product.

7.3. JS Wrapper

To test this module the following software is required:

- Node.js version 12.14.1 or higher
- NPM version 6.13.7 or higher

All test cases should be tested by writing commands to a file and executing them using the command "*node <filename>.js*".

8. Test schedule and budget

Testsuite	Begin	End
C++ Wrapper	13.04.2020	01.05.2020
JS Wrapper	13.04.2020	01.05.2020
Console Application	13.04.2020	01.05.2020

9. Test planning

Testsuite	Test objective	Test plan creator	Test plan reviewer	Tester
TS-001	C++ Wrapper	Lucas Krauter	Krister Wolfhard	Kevin Kretschmar
TS-002	JS Wrapper	Markus Limbacher	Krister Wolfhard	Joshua Franz
TS-003	Console Application	Joshua Franz	Krister Wolfhard	Markus Limbacher

10. References/Standards

[1] SRS TINF18C AMLEngine-DLL Interfaces (https://github.com/RBeerDevelopment/TINF18C_Team_4_AMLEngine-DLL-Interface/wiki/System-Requirements)

11. Appendix: Test Cases

11.1. Testsuite <TS-001 C++ Wrapper>

11.1.1. <TC-001-001> (Follow Wrapper instructions)

Testcase ID:	TC-001-001	
Testcase Name:	C++ Wrapper	
Req.-ID:	UC.001, /LF10/C++ Functions, /LF20/C++ usability	
Description:	This test case verifies that the C++ wrapper instructions are understandable and lead to a correct executable, which uses the AMLEngine.dll.	
Test Steps		
Step	Action	Expected Result
1	Install Visual Studio Community or Enterprise Microsoft.	The editor can be started.
2	Download the wrapper instructions from the official github repository of this project or open it in the browser	The wrapper documentation can be opened to read.
3	Follow the wrapper instructions	A code example using the AMLEngine.dll is ready to compile.
4	Compile the code with help of the wrapper instructions.	An executable is built by the compiler tools.
5	Run the executable.	The executable runs.

11.2. Testsuite <TS-002 JS Wrapper>

11.2.1. <TC-002-001> Wrapper import

Testcase ID:	TC-002-001	
Testcase Name:	Wrapper import	
Req.-ID:	UC.002, LF30, LF40	
Description:	Validates that the wrapper package can be downloaded and imported into a node project.	
Test Steps		
Step	Action	Expected Result
1	Make sure to have all the required software installed. (See 7.3)	n/a
2	Run the command "*npm i amlenginewrapper --save*"	Success message is displayed and the package is added to the package.json file
3	Invoke the interactive node terminal using the command "*node*"	The console displays the message "Welcome to Node.js"
4	Write the command "*wrapper=require('amlenginewrapper');*"	The package is imported without any error messages

11.2.2. <TC-002-002> Valid call and invalid call handling

Testcase ID:	TC-002-002	
Testcase Name:	Valid call and invalid call handling	
Req.-ID:	UC.002, LF30, LF40	
Description:	Validates that the wrapper package can access all supported functions inside the Adapter. Validates that the wrapper package can also handle invalid function calls.	
Test Steps		
Step	Action	Expected Result
1	Make sure to have all the required software installed. (See 7.3)	n/a
2	Run the command "*npm i amlenginewrapper --save*"	Success message is displayed and the package is added to the package.json file
3	Invoke the interactive node terminal using the command "*node*"	The console displays the message "Welcome to Node.js"
4	Write the command "*wrapper=require('amlenginewrapper');*"	The package is imported without any error messages
5	Enter the test data specified below.	The output matches the expected result.

Test data:	Inputs for call method		
	function	parameter(s)	Expected result
1	Call - instancehierarchy_append	wrapper.call("instancehierarchy_append", "pathToAMLFile", {"indexer": "indexOfHierarchyToAppendTo", "internalelement":"nameOfInternalElementToAppend"})	No errors thrown. New element is appended in the given aml file.
2	Call - instancehierarchy_get	wrapper.call("instancehierarchy_get", "pathToAMLFile", { "indexer": "indexOfInstanceHierarchyToGet"})	No exceptions thrown. Instance hierarchy is printed to the console
3	Call - create_systemunitclass	wrapper.call("create_systemunitclass", "pathToAMLFile", {"unitclasslib_name": "nameOfNewUnitClassLib", "unitclass_name": "nameOfNewUnitClass", "indexer": "indexOfInstanceHierarchy })	No errors thrown. New system unit class is stored in the given aml file.
4	Call - create_interfaceclass	wrapper.call("create_interfaceclass", "pathToAMLFile", {"interface_classname": "nameOfNewInterfaceClass", "iface_name": "nameOfNewInterface"})	No errors thrown. New interface class is stored in the given aml file.
5	Call - instanceelement_append	wrapper.call("instanceelement_append", "pathToAMLFile", {"indexer": "indexOfInstanceHierarchy", "inElement": "nameOfNewElement"	No errors. New elements are stored in the given aml file.
6	Call - rename_element	wrapper.call("rename_element", "pathToAMLFile", { "indexer": "indexOfInstanceElementToChange", "newName": "theNewNameToUse", "ie":"nameOfInternalElementT	No errors. New data is stored in the element in the given aml file.

		oRename”})	
7	Call - validate	wrapper.call(“validate”, “pathToAMLFile”, {})	The console prints problems in the aml file syntax.
8	Call - repair	wrapper.call(“repair”, “pathToAMLFile”, {})	All syntax-based problems get repaired in the aml file given.

11.2.3. <TC-002-003> Supported functions

Testcase ID:	TC-002-003	
Testcase Name:	Supported functions	
Req.-ID:	UC.002, LF30, LF40	
Description:	Validates that all supported functions can be called using their quick access option	
Test Steps		
Step	Action	Expected Result
1	Make sure to have all the required software installed. (See 7.3)	n/a
2	Run the command “*npm install AMLEngineDLLWrapper --save*”	Success message is displayed and the package is added to the package.json file
3	Invoke the interactive node terminal using the command “*node*”	The console displays the message “Welcome to Node.js”
4	Write the command “*wrapper=require('amlenginewrapper');*”	The package is imported without any error messages

5	Enter the test data specified below using the following syntax: *wrapper.<function>(<parameters>);*	The output matches the expected result.

Test data:	Supported functions		
	function	parameter(s)	Expected result
1	appendToInstanceHierarchy	wrapper.appendToInstanceHierarchy(“pathToAMLFile”, “nameOfAppend”, “internalElement”)	No errors thrown. New element is appended in the given aml file.
2	getInstanceHierarchy	wrapper.getInstanceHierarchy (“pathToAMLFile”, “indexOfInstanceHierarchyToGet”)	No exceptions thrown. Instance hierarchy is printed to the console
3	createSystemUnitClass	wrapper.createSystemUnitClass(“pathToAMLFile”, “nameOfNewUnitClassLib”, “nameOfNewUnitClass”, “indexOfInstanceHierarchy)	No errors thrown. New system unit class is stored in the given aml file.
4	createInterfaceClass	wrapper.createInterfaceClass (“pathToAMLFile”, “nameOfNewInterfaceClass”, “nameOfNewInterface”)	No errors thrown. New interface class is stored in the given aml file.
5	appendInstanceElement	wrapper.appendInstanceElement(“pathToAMLFile”, “indexOfInstanceHierarchy”, “nameOfNewElement”)	No errors. New elements are stored in the given aml file.
6	renameElement	wrapper.renameElement (“pathToAMLFile”, “indexOfInstanceElementToChange”, “theNewNameToUse”, ”nameOfInternalElementToRename”)	No errors. New data is stored in the element in the given aml file.
7	validate	wrapper.validate(“pathToAMLFile”)	The console prints problems in the aml file syntax.

8	repair	wrapper.repair(“pathToAMLFile”)	All syntax-based problems get repaired in the aml file given.

11.3. Testsuite <TS-003 Console Application>

11.3.1. <TC-003-001> UI-Test

Testcase ID:	TC-003-001	
Testcase Name:	UI-Test	
Req.-ID:	- (Usability)	
Description:	This Test Case verifies the Usability of the Console-Application	
Test Steps		
Step	Action	Expected Result
1	Download and start the Console Application in Windows 10	A Console Application should start and the Main Menu should appear
2	Check if the Main Menu is written correctly and that the UI is user friendly and understandable.	It should be user friendly, understandable and correct.
3	Check if the Options Menu is written correctly and that the UI is user friendly and understandable.	It should be user friendly, understandable and correct.
4	Check if the Validation Menu is written correctly and that the UI is user friendly and understandable. For this a File from Example Files should be verified.	It should be user friendly, understandable and correct.

5	Check if the DeCompress Menu is written correctly and that the UI is user friendly and understandable. For this a AMLX File from Example Files should be used.	It should be user friendly, understandable and correct.
6	Check if the Compress Menu is written correctly and that the UI is user friendly and understandable. For this the Files from Step 5 should be Compressed.	It should be user friendly, understandable and correct.
7	Repeat Steps 1-6 in Windows 7 if possible.	It should work the same as with Windows 10

11.3.2. <TC-003-002> Validation Test

Testcase ID:	TC-003-002	
Testcase Name:	Validation Test	
Req.-ID:	LF70	
Description:	This Test verifies the Validation Functionality of the Console Application	
Test Steps		
Step	Action	Expected Result
1	Do Step 2 and 3 with all Combinations of the Options (both off, one of the other on etc.)	
2	Validate a correct file using the validation menu.	It should validate correctly.
3	Validate an incorrect file using the validation menu. Try both options of repair error and override the old file. (Depending on the set option from the menu)	It should show an error and the option to repair it. (Depending on the set options from the menu)

11.3.4. <TC-003-003> (De-)Compression Test

Testcase ID:	TC-003-003	
Testcase Name:	(De-)Compression Test	
Req.-ID:	LF60	
Description:	This Test verifies the Compress and Decompress Functionality of the Console Application.	
Test Steps		
Step	Action	Expected Result
1	Go to the decompress menu and decompress an AMLX file (you can use the file in example files but you don't have to).	The file should be decompressed successfully.
2	Go to the compress menu and compress one file. (You can use the files from the decompression if you want to, but you don't have to)	The file should be compressed successfully.
3	Go to the compress menu and compress two or more files, but don't use a ClassModel. (You can use the files from the decompression if you want to, but you don't have to)	The files should be compressed successfully.
4	Go to the compress menu and compress two or more files and use a ClassModel. (You can use the Files from the decompression if you want to, but you don't have to)	The files should be compressed successfully.

11.3.5. <TC-003-004> Import Test

Testcase ID:	TC-003-004	
Testcase Name:	Import Test	
Req.-ID:	LF50	
Description:	This test verifies the import functionality per starting parameter of the Console Application.	
Test Steps		
Step	Action	Expected Result
1	The Console Application has the following Parameters: - path and a valid path after that - validate -> declares that the file should be validated - compress -> declares that the file should be compressed This can look like this: “ConsoleApplication.exe --path C:\File.aml -- validate” Test the functionality with a valid path and every combination (no parameters, all parameters etc.)	The Console Application should give correct errors and work correctly.
2	Test the functionality with an invalid path and every combination of parameters	The Console Application should give correct errors.